Homework #3

경기대학교 컴퓨터공학부
201511837 이상민

```
# 문제 3-1 (1,2,3)
# 정수 4 는 1, 2, 3 의 합으로 표현할 때 아래와 같이 7 가지의 방법이있다.
# 1+1+1+1, 1+1+2, 1+2+1, 2+1+1, 2+2, 1+3, 3+1 자연수 N 을 입력 받아,  N 을 1, 2, 3 의 합으로 표현할 때에,
# 몇 가지 방법이 있는지 계산하여 출력하는 프로그램을 작성하시오.(우선은 오버플로우를 고려하지 않고 알고리듬을 기술할 것.)
# 입력은 표준입력(standard input; 키보드를 통한 입력)을 사용한다.
# 입력은 첫 줄에 자연수 N 이 하나 주어진다.
# 이때, N 은 1 이상 100000 이하의 범위이다.
# 출력은 표준출력(standard output; 모니터 화면에 출력)을 사용한다.
# 주어진 자연수 N 을 1, 2, 3 의 합으로 표현할 때에 가능한 방법의 수를 정수 형태로 출력한다.
# 입력 예          출력 예
# 1               1
# 4               7
# 10              274
# 15              5768

def A(N):
    if N == 1:
        return 1
    elif N == 2:
        return 2
    elif N == 3:
        return 4
    else:
        return A(N-1)+A(N-2)+A(N-3)
N = int(input("경우의 수를 구하고 싶은 자연수 : "))
print(A(N))
```

```
# 풀이
# 변수 n을 표현 할 수 있는 경우의 수를 A(n)라 하면
# N = 4
# 1 + 3을 표현 할 수 있는 경우의 수 = A(3)
# 2 + 2를 표현 할 수 있는 경우의 수 = A(2)
# 3 + 1을 표현 할 수 있는 경우의 수 = A(1)
# A(1) = 1, A(2) = 2, A(3) = 4가 되며,
# A(4) = A(3) + A(2) + A(1) = 4 + 2 + 1 = 7 입니다.

# N = 5
# 1 + 4를 표현 할 수 있는 경우의 수 = A(4) = 7
# 2 + 3을 표현 할 수 있는 경우의 수 = A(3) = 4
# 3 + 2를 표현 할 수 있는 경우의 수 = A(2) = 2
# A(5)는 A(4) + A(3) + A(2) A(1)=0 = 7+4+2 = 13 입니다.

# 해당 알고리즘의 점화식
#      0                        if N <= 0
#      1                        if N = 1
#      2                        if N = 2
#      3                        if N = 3
#      A(N-1) + A(N-2) + A(N-3)  if N > 4

# 요약
#      이전에 만든 수에서 1,2,3을 각각을 더했을 때 현재의 수가 나오므로
#      1을 더했을 때, 2를 더했을 때, 3을 더했을 때 현재의 수가 나오는. 각각의 이전 조합의 경우의 수를 더하면 됩니다.

# 시간 복잡도 : O(n)

# 정확성 : 100% (모든 테스트 케이스 해결 가능)
```

```
# 문제 3-2(치즈 먹기)
# N * N 개의 칸이 있는 테이블이 있고, 치즈 몇 개가 테이블에 놓여있다.
# 치즈를 좋아하는 생쥐 미키는 테이블의 (1,1)에서 출발해서 (N,N)에 도착할 때까지 많은 치즈를 먹고 싶어한다.
# 하지만 머리가 나쁜 미키는 위로 올라가거나 오른쪽으로만 이동할 수 있다.
# 예를 들어, 아래 그림에서 미키는 5 개의 치즈 중 최대 3 개의 치즈만 먹으며 이동할 수 있다.
# 테이블 크기 N 과 M 개의 치즈 위치가 주어졌을 때 최대로 먹을 수 있는 치즈의 개수를 구하는 프로그램을 작성하시오.
# 치즈의 위치는 (x,y)의 좌표로 주어지며 왼쪽 아래 구석의 위치를 (1,1)로, 맨 오른쪽 위 구석의 위치를 (N,N)으로 한다.
# 입력
# 입력은 표준입력(standard input; 키보드를 통한 입력)을 사용한다.
# 입력은 첫 줄에 자 연수 N 과 M 이 주어진다.
# 이 때, N 과 M 은 1 이상 10000 이하의 범위이다.
# 다음 M 개의 줄에 각각의 치즈의 위치 x 와 y 가 주어진다.
# 출력
# 출력은 표준출력(standard output; 모니터 화면에 출력)을 사용한다.
# 주어진 입력에 대해 최대로 먹을 수 있는 치즈의 개수를 정수 형태로 출력한다.
# 설명
# 3 = 3x3 행렬 2 = 치즈의 개수
# 1 2 (치즈의 좌표), 3 1(치즈의 좌표)
# 입력 예  입력 예에 대한 출력
# 3 2           1
# 3 2
# 1 2
# 3 1

# 5 5           3
# 2 3
# 3 2
# 4 3
# 4 5
# 5 2
```

```python
def Way(i, n):                                          # i, i 부터 i, n 까지 and i, i 부터 n, i 까지 치즈의 최대값을 구한다
    E[i][i] = max(E[i-1][i], E[i][i-1]) + T[i][i]       # (i, i)에서 먹을 수 있는 치즈의 최댓값 = 왼쪽 좌표, 아래 좌표 중 큰 값 +
                                                        # (i, i)위치의 치즈 개수
    if i == n:                                          # 2 차원 배열의 마지막에 도달하였다면,
        return E[n][n]                                  # n, n 에서 먹을 수 있는 치즈의 최댓값 리턴
    else:                                               #
        for j in range (i+1, n+1):                      # (i+1, i)부터 (n, i) / (i, i+1)부터 (i, n)까지 치즈값
            E[i][j] = max(E[i-1][j], E[i][j-1]) + T[i][j]   # 왼쪽에서 온 경로의 값이 더 큰지, 아래에서 온 경로의 값이 더 큰지 비교 후
            E[j][i] = max(E[j-1][i], E[j][i-1]) + T[j][i]   # 더 큰 값에 현재 좌표의 위치에 존재하는 치즈의 개수를 더한다.
        return Way(i+1, n)                              # i 를 1 씩 증가하여 반복한다. 테이블의 크기가 N*N 다음 N-1*N-1,
                                                        # N-2*N-2,,,로 줄어들며 N,N 에 도착하게되면 종료한다.
N, M = map(int, input("").split(' '))                   # N(치즈 테이블의 크기), M(치즈의 개수)
T = [[0 for col in range(N+1)]for row in range(N+1)]    # 배열 초기화 & 생성
E = [[0 for col in range(N+1)]for row in range(N+1)]    # 배열 초기화 & 생성
for i in range (M):                                     # 치즈의 좌표를 모두 설정하기 위해 치즈의 개수만큼 반복
    Q, W = map(int, input("").split(' '))               # Q, W(치즈의 좌표를 입력)
    T[Q][W] = int(1)                                    # 치즈의 좌프를 1 로 설정.
print('개수: ',Way(1, N))                               # 위에서 선언한 함수를 사용하여서 치즈를 최대로 먹을 수 있는 개수를 구한다.
# 해당 알고리즘의 점화식
#       -1                                              if i, j < 0
#       0                                               if i, j = 0 (치즈가 없는 경우)
#       1                                               if i, j = 1 (치즈가 있는 경우)
#       E[i][j] = max(E[i-1][j], E[i][j-1]) + T[i][j]   if i, j > 1
#       E[j][i] = max(E[j-1][i], E[j][i-1]) + T[j][i]
# 요약
#       해당 알고리즘을 요약하자면, 테이블의 크기를 줄여가면서 이전 경로에서의 값을 비교하여서 2 개의 경로 중 최대의 값을 가지는 경우를
#       계속해서 더해가면서 목적지에 도착하게 되면 목적지에서의 총 합계를 반환하는 알고리즘입니다.
# 시간 복잡도 : O(n^2)
# 정확성 : 100% (모든 테스트 케이스 해결 가능)
```

```python
# 문제 3-3 (동전)
# Nadiria 라는 (상상의) 나라에서는 다음과 같은 액면가의 동전(화폐)을 사용한다고 한다.
# $1, $4, $7, $13, $28, $52, $91, $365
# 어떤 곳이든 사람들은 돈을 주고 받을 때, 가능한 적은 개수의 동전을 사용하기를 원한다.
# 입력으로 자연수 K 가 주어지면 Nadiria 화폐를 이용하여 $K 를 만들 수 있는 최소 동전 개수를 출력하는 알고리즘을 설계하고 분석하시오.

Coins = [1,4,7,13,28,52,91,365]
def Nadiria(Coins, Money):
    arr = [0] * (Money + 1)                     # 0 의 배열을 Money 의 개수만큼 생성.
    for i in range(1, Money + 1):               # 1 부터 Money 까지 탐색.
        Temp = 9999                             # 첫번째 원소를 무조건 반환하기 위해서 사용하지 않을만한 큰 값을 설정.
        j = 0                                   # 화폐 종류를 모두 보기 위해서 0 으로 초기화.
        while j < len(Coins) and i >= Coins[j]: # 화폐의 종류만큼 반복하고, Money 의 값이 화폐보다 크면 값을 비교한다.
            Temp = min(arr[i-Coins[j]], Temp)   # 현재의 Money 에서 화폐의 단위를 뺀 최소의 화폐 개수를 저장
            j += 1                              # 화폐의 종류 만큼 반복하기 위해서 j 를 1 씩 증가.
        arr[i] = Temp + 1                       # 현재의 머니에서 화폐 단위를 뺀 최소 개수에 +1 하여 저장.
    return arr[Money]
print(Nadiria(Coins,600))

# 점화식
#       0                               if Money <= 0
#       min(arr[i-Coins[j]], Temp)      if Money > 0

# 요약
#       화폐의 크기만큼 배열을 선언 후 최종 목표하는 금액을 0 부터 시작하여 계산된 최소 동전의 수를 이용하여서
#       목표하고자하는 금액의 최소 화폐의 개수를 구하는 알고리즘입니다.

# 시간 복잡도 : O(n)

# 정확성 : 100% (모든 테스트 케이스 해결 가능)
```

```python
# 문제 3-4 (최대 합 부분 배열)
# 길이가 n 인 정수의 배열 A[0..n-1]가 있다.
# A[a] + A[a+1] + ... + A[b]의 값을 최대화하는 구간 (a, b)를 O(n) 시간 안에 찾는 방법을 설계하고 분석하라.
# 예를 들어, 배열 A 가 아래와 같이 주어졌을 경우 (n = 10), 31 -41 59 26 -53 58 97 -93 -23 84
# 답은 a = 2, b = 6 인 경우의 59+26-53+58+97=187 가 된다.

A = [31, -41, 59, 26, -53, 58, 97, -93, -23, 84]
def MaxSumArray(A):
    if len(A) <= 0:                         # 배열의 길이가 0 이하일 경우 -1 리턴하여 예외 처리.
        return -1
    Temp = [None] * len(A)                  # 구하고자하는 배열 A 의 길이 만큼 새로운 배열을 생성.
    Temp[0] = A[0]                          # 새로운 배열의 첫번째 원소는 배열 A 의 원소를 복사.
    for i in range(1, len(A)):              # 배열의 인덱스 번호 1 부터 입력한 배열의 끝까지 반복.
        Temp[i] = max(0, Temp[i-1]) + A[i]  # Temp 배열의 A 의 원소를 더해가면서 진행. Temp 의 이전 원소가 음수라면 0 으로 초기화하고 A 의
                                            # 원소를 더한다.
        if (Temp[i-1] < 0):                 #
            a = i                           # Temp 의 원소가 -가 나오면 해당 인덱스의 다음 인덱스 정보가 시작 인덱스 번호.
    print("a:",a)                           # 구간 a 출력.
    print("b:",Temp.index(max(Temp)))       # 구간 b 출력. 최대값이 존재하는 원소의 인덱스 번호
    return max(Temp)                        #
print(MaxSumArray(A))                       #

# 점화식
#       -1                          if 배열의 길이 <= 0
#       max(0, Temp[i-1]) + A[i]    if 배열의 길이 > 1
# 요약
#       해당 알고리즘을 요약하자면, 문제에 주어진 배열을 탐색하면서 이전의 배열의 원소와 이후의 배열의 원소를 더하여 새로운 배열에 저장을 하는 과정을
#       반복하고 이전의 결과물을 다음의 결과에 사용하면서 최대 합의 구간을 찾아내는 알고리즘입니다.
# 시간 복잡도 : O(n)
# 정확성 : 100% (모든 테스트 케이스 해결 가능)
```

```
# 문제 3-5(최대 곱 부분 배열)
# 길이가 n 인 정수의 배열 A[0..n-1]가 있다.
# A[a]*A[a+1] * ... * A[b]의 값을 최대화하는 구간 (a, b)를 찾는 방법을 설계하고 분석하라.
# 배열 A 의 원소는 양수, 음수,0 모두 가능하다.
# 예를 들어, 배열 A 가 아래와 같이 주어졌을 경우 (n=7), -6 12 -7 0 14 -7 5
# 답은 a=0, b=2 인 경우의 (-6)*12*(-7)=504 가 된다.

A = [-6, 12, -7, 0, 14, -7, 5]                          # 대상이 되는 배열
X = [0 for i in range(len(A))]                           # 배열 A 의 길이만큼 배열 X 를 0 으로 채워 생성.
X[-1] = A[-1]                                            # 배열 X 의 끝을 배열 A 의 끝으로 선언.
def MaxProdArray(A):                                     #
  MinT = [0 for i in range(len(A))]                      # 배열 A 의 길이만큼 배열 MinT 를 0 으로 채워 생성.
  MaxT = [0 for i in range(len(A))]                      # 배열 A 의 길이만큼 배열 MaxT 를 0 으로 채워 생성.
  MinT[0] = A[0]                                         # MinT, MaxT 의 첫 원소를 A[0]으로 초기화.
  MaxT[0] = A[0]                                         #
  a, b = 0, 0                                            # 최대곱 구간을 저장하기 위한 변수 선언.
  for j in range(1,len(A)):                              # A[0]의 값을 이미 넣어두었기 때문에 A[1]을 제외한 반복.
    MinT[j] = min(MinT[j-1]*A[j], MaxT[j-1]*A[j], A[j])  # MinT 의 이전 원소와 대상이 되는 A 원소의 곱의 결과, MaxT 의 이전 원소와
                                                         # 대상이 되는 A 원소의 곱의 결과,
    MaxT[j] = max(MinT[j-1]*A[j], MaxT[j-1]*A[j], A[j])  # 대상이 되는 A 원소의 값, 세 가지를 비교하여서 가장 큰 값과 가장 작은 값을
                                                         # 저장하여 다음 인덱스를 사용하여 반복을 진행.
  if (max(MaxT) < max(MinT)):                            # 가장 큰 값이 MinT 에 있는 경우.
    b = MinT.index(max(MinT))                            # 최대 값의 구간의 끝을 구하기 위해 MinT 에서 가장 큰 값의 인덱스 번호를 b 로
                                                         # 넘긴다.
  else:                                                  # 가장 큰 값이 MaxT 에 있는 경우.
    b = MaxT.index(max(MaxT))                            # 최대 값의 구간의 끝이 MaxT 에 존재하여 MaxT 에서 가장 큰 값의 인덱스 번호를
                                                         # b 로 넘긴다.
  for j in range(len(A)-1,0,-1):                         # 최대 곱의 시작 구간을 찾기 위한 반복
```

```
    X[j-1] = X[j]*A[j-1]                                    # A 배열의 끝부터 시작하여 인덱스 번호 0 까지 반복하며, 배열의 길이의 -1 만큼
                                                            # 반복.
    if (X[j] == max(MaxT)):                                 # X 에 저장된 최대 값의 값과 MaxT 의 존재하는 최대 값과 일치한다면
      a = j                                                 # X 에 해당하는 인덱스 번호를 시작점으로 선언.
  print("a:", a, "b:", b)                                   # 최대 곱의 구간 출력
  return max(MinT,MaxT)
print(max(MaxProdArray(A)))

# 점화식
# -1                                                    if 배열의 길이 <= 0
# MinT[j] = min(MinT[j-1]*A[j], MaxT[j-1]*A[j], A[j])   if 배열의 길이 > 1
# MaxT[j] = max(MinT[j-1]*A[j], MaxT[j-1]*A[j], A[j])

# 요약
# 배열 하나를 입력으로 받아 최대 곱을 저장하는 배열, 최소 값을 저장하는 배열 두개를 생성 후 두 개의 배열 결과를 가지고 비교하여
# 새로운 배열에 최대 값을 넣어 최대 곱의 구간의 범위를 알아내는 알고리즘입니다.

# 시간 복잡도 : O(n)
# 정확성 : 100% (모든 테스트 케이스 해결 가능)
# 해당 문제는 임한민, 정범식, 한상준 학우와 함께 작성 후 풀이를 각자 진행하였습니다.
```